President Trump Tweets generation Using RNN

Suhas Pavagada Shekhar

Abstract— Natural Language Processing (NLP) is a hot research topic in deep learning. In this project, I am doing Language modeling for text generation. I am using Recurrent Neural Network to model both syntactic and semantic information of tweets. Neural networks have shown that it can capture the text features for many applications. There are so many open source projects on generating tweets. In the report, I will discuss the different types of architecture of RNN and different varieties to model the text used in these projects. All these projects make use of an LSTM model that hierarchically builds an embedding for tweets from embeddings for words and characters, then decodes this embedding to reconstruct the original tweets. RNN architecture is a probabilistic determination model. The tweets are generated using Log prediction of words using current step tokens and the predicted token from the earlier step. In the end, I will discuss the different evaluation metric for the generated tweet showing that neural models are able to encode texts in a way that preserve syntactic, semantic, and discourse coherence.

Index Terms— NLP - Natural Language Processing, RNN - Recurrent Neural Network. LSTM- Long Short Term Memory, GAN- Generative Adversarial Network.

INTRODUCTION

Generating text which is both syntactically and semantically meaningful is a difficult process in NLP. There are many types of theories which tried to model the unit text representation with each other in a sentence, such as Rhetorical Structure Theory (Mann and Thompson, 1988) or Discourse Representation Theory (Lascarides and Asher, 1991), for extracting these relations from text units (Marcu, 2000; LeThanh et al., 2004; Hernault et al., 2010; Feng and Hirst, 2012, inter alia), and for extracting other coherence properties characterizing the role each text unit plays with others in a discourse (Barzilay and Lapata, 2008; Barzilay and Lee). Even with the availability of these resources, applying these to text generation is still difficult. To understand how text units are connected, one must understand the communicative function of each unit, and the role it plays within the context that considers of all other units. As the ways of communication between humans are changing our language features and models as well. Finding these increasingly sophisticated human-developed features are insufficient for the classical theories. The recent development in Deep Learning showed a way for new theories to model the natural language.

The first attempt in modeling text using probabilistic prediction using Neural network "Generating text with recurrent neural networks" (I Sutskever, J Martens, GE Hinton, 2011) showed the way for modern NLP. Ever since RNN have shown promising results with the most common approach is to maximize the predictive likelihood of each true token in the training sequence given the previously observed tokens. Recent LSTM models (Hochreiter and Schmidhuber, 1997) have shown powerful results on generating meaningful and grammatical sentences.

After the introduction of Generative Adversarial Network (GAN), NLP community has made improvements in the generation task. Generative Adversarial Network has overcome some of the drawbacks of RNN. When we try to generate sentences from RNN architecture, the error will accumulate exponentially with the length of the sentence. The first several words can be relatively

reasonable; however, the quality of sentence deteriorates quickly. In addition, the lengths of sentences generated from random latent representations could be difficult to control. The recently published paper" Evaluating Generative Models for Text Generation" has shown improved results over RNN. In this paper, they have evaluated two different methods using GAN, Scheduled sampling (Bengio et al. (2015)) and SeqGAN, Yu et al. (2016). Both models use reinforcement learning with the traditional approach.

For this project, I have explored many ways to do the task of RNN natural language generation. RNNs are simple, powerful, robust and easy to implement. RNN can be very diverse, sequential input to sequential output (eg. Machine translation), sequential input to single output (eg. Classification), single input to sequential output (eg. Image captioning).

The focus was on the component task of training tweets (words of 140 to 240 characters) autoencoder to reconstruct the input text from a vector representation of words and characters from a deep learning model. I have followed many open source projects form pioneers in NLP for this project. I have implemented hierarchical LSTM models that arranges tokens of characters and words in a hierarchical structure, with different levels of LSTMs.

In the following section, a brief description of RNN and LSTM is discussed. The proposed hierarchical LSTM models are then described in Section 3, followed by experimental
results in Section 4, and then a brief conclusion.

PROCEDURES

In this section, we will discuss the methods, architecture of the models and datasets used for training and evaluation.

Methods

The main task of this project is text generation. I make use of RNN with LSTM units for this task. Let's have a quick overview of LSTM.

Long Short Term Memory

LSTM models (Hochreiter and Schmidhuber, 1997) are defined as follows: given a sequence of inputs, an $X = \{x_{1,}x_{2,} \dots \dots . x_{n_x,}\}$ LSTM associates each timestep with an input, memory and output gate, respectively denoted as i_t, f_t and o_t.

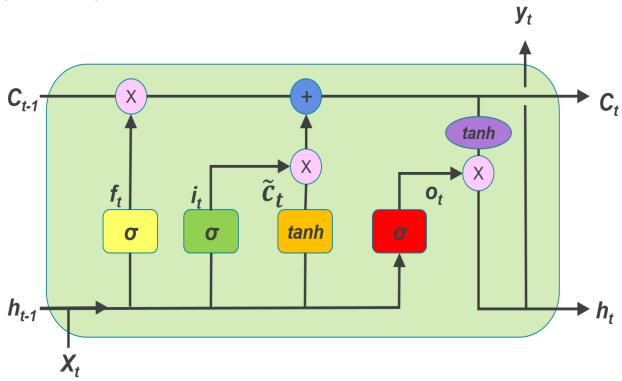

Figure 1. Architecture of a LSTM

The model is given by

$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ l_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} W \cdot \begin{bmatrix} h_{t-1} \\ e_t \end{bmatrix}$$

$$c_t = f_t \cdot c_{t-1} + i_t \cdot l_t$$

$$h_t^s = o_t \cdot c_t$$

$\boldsymbol{e_t}$ denote the vector for individual text unite (e.g., word or sentence) at time step $\boldsymbol{t}$. $\boldsymbol{h_t}$ denotes the vector computed by LSTM model at time t by combining $\boldsymbol{e_t}$ and $\boldsymbol{h_{t-1}}$. $\boldsymbol{\sigma}$ denotes the sigmoid function and $\boldsymbol{W} \in \mathbb{R}^{\boldsymbol{4K \times 2K}}$.
In sequence-to-sequence generation tasks, each input X is paired with a sequence of outputs to predict: $Y = \{y_{1,}y_{2,} \dots \dots . y_{n_Y,}\}$. An LSTM defines a distribution over outputs and sequentially predicts tokens using a softmax function:

$$\boldsymbol{P(X|Y)} = \prod_{\boldsymbol{t\epsilon[1,n_y]}} \boldsymbol{p(y_t|x_1, x_2, \cdots, x_t, y_1, y_2, \cdots, y_{t-1})}$$

$$\boldsymbol{P(X|Y)} = \prod_{\boldsymbol{t\epsilon[1,ny]}} \frac{\boldsymbol{exp\ (f(h_{t-1}, e_{yt}))}}{\boldsymbol{\sum_{y'} exp\ (f(h_{t-1}, e_{y'}))}}$$

Where $f(h_{t-1}, e_{yt})$ denotes activation function and h_{t-1} represents output from LSTM at $t-1$. that

each sentence ends up with a special end-of sentence symbol <end>. Commonly, the input and output use two different LSTMs with different sets of convolutional parameters for capturing different compositional patterns. In the decoding procedure, the algorithm terminates when an <end> token is predicted. At each timestep, either a greedy approach or beam search can be adopted for word prediction. Greedy search selects the token with the largest conditional probability, the embedding of which is then combined with preceding output for next step token prediction.

Autoencoders

An autoencoder is a neural network that is trained to attempt to copy its input to its output. Internally, it has a hidden layer that maps the input to the output. The network may be viewed as consisting of two parts: an encoder and a decoder. This architecture is presented in figure. Autoencoders are designed to learn to copy its input perfectly. Usually, they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data. Because the model is forced to prioritize which aspects of the input should be copied, it often learns useful properties of the data.

Earlier, autoencoders were used for dimensionality reduction or feature learning. Recent development in deep learning has brought autoencoders and latent variable models to the forefront of generative modeling.

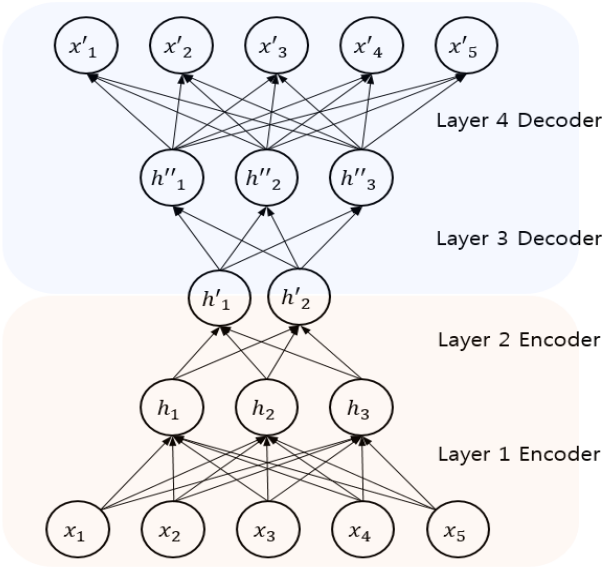

Figure 2. Autoencoder

Models

There are many variants of RNN to achieve the task of text generation, such as

- Character Based model
- Word Based model
  (I Sutskever, J Martens, GE Hinton 2011)
- Hierarchical Autoencoder
  (Jiwei Li, Minh-Thang Luong, Dan Jurafsky)
- Attention Hierarchical Autoencoder
  (Jiwei Li, Minh-Thang Luong, Dan Jurafsky)
- Variation Autoencoder
- Bidirectional LSTM

All these models vary in complexity of implementation. They have their own uniqueness. For the project, I have worked on two models Character based model and Word-based model. In the given time, these two models worked best with the simplicity of implementation.

Notation

Let T denotes a tweet, which at most has two to three sentences denoted by N_D , $T = \{s^1, s^2, \dots\dots, s^{N_D}, end_D\}$. Each tweet is represented as vector representation $e_{D.}$ s denotes each sentence, which as sequence of N_S words, s= $\{w^1, w^2, \dots\dots, w^{N_s}, end_s\}$. Each sentences s is associated with a K-dimensional embedding e_s. Each word w is associated with K-dimensional embedding $e_w = \{{e_w}^1, {e_w}^2, \dots\dots, {e_w}^K\}$.

V denotes vocabulary size. LSTM(h_{t-1}, e_t) to be the LSTM operation on vectors h_{t-1} *and* e_t to achieve h_t. h_t^s and h_t^w denote hidden vectors from LSTM models. $h_t^s(enc)$ specifies encoding stage and $h_t^s(dec)$ specifies decoding stage. e_t^w and e_t^s denotes word-level and sentence level embedding for word and sentence at position t.

An autoencoder is a neural model where output units are directly connected with or identical to input units. Typically, inputs are compressed into a representation using neural models (encoding), which is then used to reconstruct it back (decoding). For a paragraph autoencoder, both the input X and output Y are the same document D. The

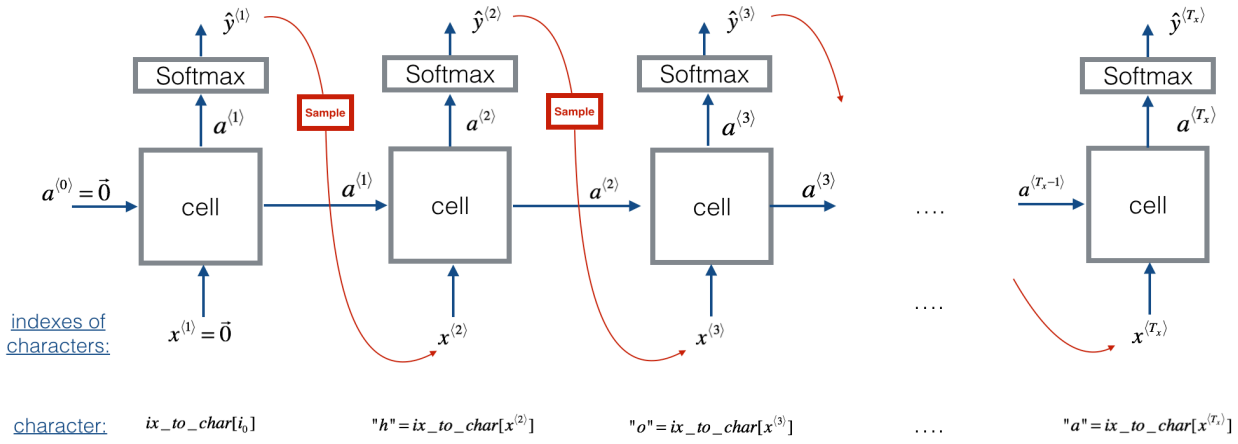

Figure 3. Architecture of Character level RNN

Architecture

Character Based Model

Character models are the easiest to implement because of its simplicity. There is no need for a large vocabulary of all the words in the corpus. We don't need to have a large word embedding for each token when compared to the word based model. Since we want to train RNN character-level language models, the One-hot Representation is the most common way for our project. This approach encodes each character into a vector using a 1-of-k encoding which means there are all zero except for a single one at the index of the character in the vocabulary. For example, the character "a" is encoded as [1 0 0 0 ...]. Each character is fed into the encoder and decoder layer at each time step in the forward pass. This process continues until the system encounters <end> token. At each time the input token gets compared with the target label, which is nothing but the character in the next time step. The error gets accumulated until the completion of the forward pass. At the backward propagation, the total error is minimized for the given weight vectors and for the process of backpropagation continues until <sos> token. This process keeps continuing for all the samples of tweets chosen for the given duration of training.

Word Based Model

In this model, we need to train our neural net on word level embedding. We go through all the words in the data corpus and add the unique words to the vocabulary. The number of unique words constitutes the size of our vocabulary. The language model will be statistical and will predict the probability of each word given an input sequence of text. The predicted word will be fed in as input to in turn generate the next word. A key design decision is how long the input sequences should be. They need to be long enough to allow the model to learn the context of the words to predict. This input length will also define the length of seed text used to generate new sequences when we use the model. The first thing we have to take care of is how we define the input to our language modeling function. The input is a sequence of words, but, how each of these words should be represented? As in most deep learning systems, we would like to constraint the model with prior knowledge as least as possible. One encoding scheme that meets this criterion is a 1-of-K encoding (one-hot-encoding). Under this scheme, each word i in the vocabulary V is represented as a binary vector w_i. To denote the i^{th} word with the vector w_i, we set the i^{th} element of it to be 1 and all the other elements to be 0.

$$w_i = [0,0,\ldots,1(i^{th}\text{ element}),0,\ldots,0]^T \in \{0,1\}^V$$

The prior knowledge embedded in this encoding scheme is minimal in the sense that the distance between any two word vectors is equal to 1 if the two words are different and 0 if the words are the same word.

Now, the input to our model is a sequence of T−1 one hot vectors $(w_1, w_2, \ldots\ldots, w_{T-1})$. Each of these vectors will be then multiplied by a weight matrix E, to get a sequence of continuous vectors $(x_1, x_2, \ldots\ldots, x_{T-1})$ such that,

$$x_j = E^T w_j$$

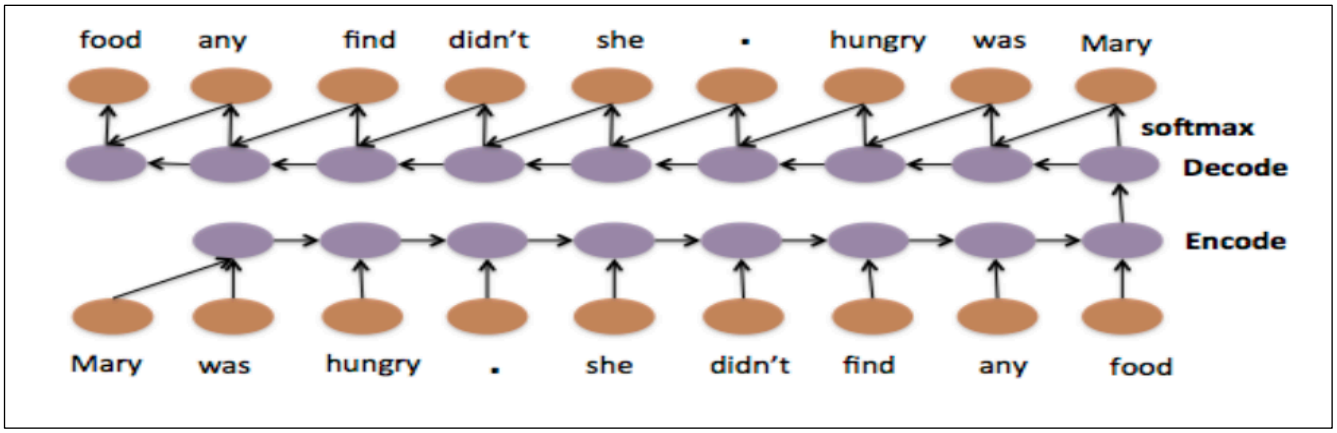

Figure 4. Architecture of Word level RNN

The dimension of matrix E is $200 \times V$ .Each word is fed into the encoder and decoder layer at each time step in the forward pass. This process continues until the system encounters <end> token. At each time the input token gets compared with the target label, which is nothing but the character in the next time step. The error gets accumulated until the completion of the forward pass. At the backward propagation, the total error is minimized for the given weight vectors and for the process of backpropagation continues until <sos> token. This process keeps continuing for all the samples of tweets chosen for the given duration of training.

Data Preprocessing

The dataset consists of Trump tweets from 2009. The dataset is collected from a lovely website http://www.trumptwitterarchive.com. The total tweets in the data set is roughly 32000.

To create word embedding for the data is very time consuming. I made use of pretrained word embeddings called GloVE Word Library. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. The resulting representations showcase interesting linear substructures of the word vector space. It is trained over different dataset for generating word embedding. For the project I made use of Twitter word embedding consists of 2B tweets, 27B tokens, 1.2M vocab, uncased and has different dimensional word embedding size 25d, 50d, 100d, & 200d vectors.

To make use of GloVe I cleaned all the tweets for symbols attached to the words to reduce the number of <UNK> token in the vocabulary (eg: @realDonoldTrump; @ realDonoldTrump) and https address. This reduces the <UNK> in the vocabulary from 31% to 9%. The total of 10000 tweets were used for training. The Size of Vocabulary is 22931 words. Glove Word embeddings as vocabulary size of 1193514.

Training

Character Based Model

The architecture for training consists of four hidden layers. Two for each encoder and decoder layers. Each layer has 512 LSTM units. To train the model I have used the cross entropy as a loss function.

$$E = -y * log(sigmoid(\hat{y})) - (1 - y) * log(1 - sigmoid(\hat{y}))$$

The objective is to minimize the error between input and output probability distribution. The last layer is linear layer with 98 units which is followed by SoftMax activation function. The ADAM stochastic gradient descent method is used as an optimizer. The learning rate is set at 0.001. The batch size of 512 samples were used for training. The tweet of max length of 60 words are used. The dropout value of 0.2 is used to overcome overfitting. The model was trained for 20 epochs.

Word Based Model

The architecture for training consists of four hidden layers. Each encoder and decoder have two layers of 512 hidden LSTM units. I have trained the

model using the Cross entropy. The last layer is linear layer of size equal to the vocabulary size, which is followed by SoftMax activation function. The ADAM stochastic gradient descent method is used as an optimizer. The learning rate is set at 0.005. The batch size of 32 samples were used for training. We initialize the weights for all hidden layers with a normal distribution of zero means. The dropout value of 0.5 is used. The model was trained for 59 epochs. I trained the model on 10000 tweets. Total Number of Unique Words in tweets 15314. Total Number of Words: 200419. The predicted char length is cut off at 50 words.

Results

There is no general evaluation metric to check the quality of the generated text, because of the difficulty of developing a universal coherence evaluation metric which suits for all kind of NLP. Typically ROUGE and BLEU scores are used to define the coherent characteristics of text. The results were tabulated in below table.

Model	ROUGE_1	BLEU_1
Character Based	**0.62**	**0.54**
Word Based	**0.72**	**0.61**

Table 1. Unigram Score

Model	ROUGE_2	BLEU_2
Character Based	**0.31**	**0.24**
Word Based	**0.37**	**0.3**

Table 2. Bigram Score

CONCLUSIONS

We can clearly conclude that word based model outperformed the character based model. But the predicted tweets from the word based model are not clearly same as the real tweets. It is very far from the actual. Character tweets looks like real tweets because of its simplicity in modelling each character. We can clear see the format of a website, a retweet, date and hashtags, but doesn't make sense.

Word-based LMs display higher accuracy and lower computational cost than char-based LMs. This drop in performance is unlikely due to the difficulty for character level model to capture longer short term memory, since also the LSTM recurrent networks work better with word-based input. This is because char-based RNN LMs require much bigger hidden layer to successfully model long-term dependencies which mean higher computational costs. Therefore, we can say that one of the fundamental differences between the word level and character level models is in the number of parameters the RNN has to access during the training and test. The smaller is the input and output layer of RNN, the larger needs to be the fully connected hidden layer, which makes the training of the model expensive. Char-based RNN LMs can mimic grammatically correct sequences for a wide range of languages, require bigger hidden layer and computationally more expensive while word-based RNN LMs train faster and generate more coherent texts and yet even these generated texts are far from making actual sense.

Future Work

The RNN models used here are very basic. We can surely get better results with the use of more complex models mentioned in the earlier sections. Rather than using pre-trained models for word embedding, I can create my own embeddings. I will be time-consuming but tailor-made for the particular dataset so the results will be better. Generative models have shown better results than RNN in Machine Translation and other applications. So making use of GANs give the better quality of the generated text. I surely hope to explore all these new architecture and complex models in the future. We can also make use this architecture for different applications such as sentiment analysis and classification as well.

Generated Tweets

Word Models

- @ bobby990r1 mediabiasalert @ megynkelly @ realdonaldtrump is fighting for middle
- i am not a fan of the people who have been in the
- @ oreillyfactor please correct i won virginia ! i am going to make
- @ joemarzocco realdonaldtrump they attack you to show you how
- why would the ridiculous campaign on the fbi
- @ geraldorivera @ sentedcruz got ass kicked by @ the great billy

- i will be interviewed on @ oreillyfactor tonight at 8 : 00 pm eastern
- @ leaving : north korea : another rally at the @ whitehouse # s544
- @ drudgereport : trump wins cnbc instant poll rubio second developing
- @ club4growth has been allowed to do much better
- campaign has been in the path of the fact that i have spent a
- news @ cnn national poll released thank you
- rt @ rightly news :
- thank you @ dallaspd ! # makeamericagreatagain # trump2016 @ fox news is
- @ carlyfiorina is a total phony
- obama has been largely forgotten
- @ fox news is so biased and
- lyin' ted cruz and 1 for 38 kasich are a total phony
- i am watching my campaign in the house to vote for
- to be in dallas tonight at the @ whitehouse today
- thank you new hampshire for your support ! i

REFERENCES

[1] Sepp Hochreiter and J¨urgen Schmidhuber. 1997. Long short-term memory. Neural computation , 9(8):1735–1780.

[2] Mirella Lapata and Regina Barzilay. 2005. Automatic evaluation of text coherence: Models and representations. In IJCAI , volume 5, pages 1085–1090.

[3] Ilya Sutskever, Oriol Vinyals, and Quoc VV Le. 2014. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems , pages 3104–3112.

[4] Z. Shi, M. Shi and C. Li, "The prediction of character based on recurrent neural network language model," 2017 IEEE/ACIS 16th International Conference on Computer and Information Science.

[5] S. Salekin, J. Zhang, Y. Huang ”A deep learning model for predicting transcription factor binding location at Single Nucleotide Resolution” 2017 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI) Conference

[6] A Hierarchical Neural Autoencoder for Paragraphs and Documents Jiwei Li, Minh-Thang Luong and Dan Jurafsky

[7] Generating text with recurrent neural networks I Sutskever, J Martens, GE Hinton - Proceedings of the 28th International Conference …, 2011

[8] Evaluating Generative Models for Text Generation, Prasad Kawthekar, Raunaq Rewari, Suvrat Bhooshan

[9] http://karpathy.github.io/2015/05/21/rnn-effectiveness

[10] https://www.deeplearningbook.org/contents/autoencoders.html

[11] https://towardsdatascience.com/tweet-like-trump-with-a-one2seq-model-cb1461f9d54c